Advanced Legion Features

Mike Bauer
Managing Mapped Regions

- Runtime API is detailed
  - Manage both logical and physical regions
  - Similar to managing variables and registers
  - Legion language doesn’t have this problem

- Runtime injects unmap and map operations to avoid deadlocks
  - Better for applications to do this themselves
  - `unmap_all_regions()`

```c
task parent(region R) {
    unmap_all_regions();
    launch child task : R
    launch child task : R
}
```
Virtual Mappings

- Virtual mappings: don’t make an instance!
  - Pass privileges only
  - Create instances only where they are actually needed

- More detailed mapping information
  - Slightly more expensive meta-analysis

![Diagram of virtual mappings]

http://legion.stanford.edu
Tunable Variables

What about variables that depend on some aspect of the machine?
- Circuit: how many pieces?

Answer: tunable variables
- Defer decision to the mapper at runtime
- Mapper picks based on introspection of machine

get_tunable_value(...)
- Currently just integers
- Anything other types?
Relaxed Coherence Modes

- Default coherence mode is Exclusive
  - Guarantees program order execution of tasks

  \[ t_1: \text{Read-Write} \text{ Exclusive} r \]

  \[ t_2: \text{Read-Write} \text{ Exclusive} r \]

- In some cases Exclusive is overly restrictive
  - What if we just need serializability?
  - What if we want to do our own fine-grained synchronization?

- Solution: relaxed coherence modes
Atomic Coherence

- Guarantee serializable access to logical regions
- Runtime can re-order tasks

\[ t_1: \text{Read-Write } \textbf{Exclusive} \ r_1, \text{Read-Write } \textbf{Atomic} \ r_2 \]

\[ t_2: \text{Read-Write } \textbf{Atomic} \ r_2 \]

- Currently implemented using reservations
- Could also use transactional memory in the future
Simultaneous Coherence

- Tasks can run concurrently even if both writing
  - Tell Legion: Don’t worry, I’ve got this 😊
  - Makes no guarantee of concurrent execution

\[ t_1: \text{Read-Write Simultaneous } r \]

\[ t_2: \text{Read-Write Simultaneous } r \]

- Ensure that all updates are observed by all tasks
- Application responsible for synchronization
  - Reservations and phase barriers

http://legion.stanford.edu
Stencil Computation

Example: 1-D stencil
- Region of data
- Two fields: input, output
- 5 point stencil

Need 2 nearest neighbors on each side to compute stencil
Implicit Ghost Cells

- **Standard Legion way**
  - Multiple Partitions
    - Owned cells
    - Ghost cells
  - Index Space Task Launch

- **Legion computes dependences**
  - Handles movement of data and meta-data
Explicit Ghost Cells

- Have parallel running tasks (SPMD) that exchange data through explicit ghost regions

- Tasks request privileges on owned+ghost regions
  - Owned cells
  - Owned ghost cells
  - Neighbor ghost cells

- Don’t these overlap?
  - Yes!
  - Simultaneous Coherence

http://legion.stanford.edu
Phase Barriers

- Problem: Legion cannot detect dependences between different contexts
  - How do we synchronize?
  - Answer: phase barriers

- These are not MPI barriers

- Producer-consumer sync.
  - Set of arrivals
  - Set of waiters
  - Both sets can be dynamically computed
  - Launchers have entries for arriving and waiting

Region-to-Region Copies

December 4, 2014  
http://legion.stanford.edu
Must Epoch Launches

Problem: how do we guarantee that SPMD tasks can synchronize with each other using phase barriers?

- Legion makes no guarantee about concurrent execution
- Answer: must epoch launches

Must epoch launchers are meta-launchers

- Containers for normal launchers: single and index space
- Declarative way of saying tasks must all run concurrently

Can use any coherence

- Legion checks for interference between tasks
- Reports errors if tasks cannot be run concurrently
Mapping Must Epoch Launches

- Must epoch launches place constraints on mappers
  - Must map each task to a different processor
  - All interfering regions with simultaneous coherence must map to the same physical instance

- Separate mapper interface call
  - virtual void map_must_epoch(...)  
  - Map all tasks at the same time
  - Given set of simultaneous constraints to be satisfied
  - Runtime checks that all tasks can run concurrently
    - Otherwise mapping fails
Restricted Access

Simultaneous restriction
- All writes need to be immediately visible
- Runtime cannot freely make copies
- Mark restricted field on region requirements

Can we map stencil task on GPU? (not currently)
- Explicit ghost cell regions mapped to CPU-DRAM Memory
- Not visible on GPU
Acquire and Release Operations

- Acquire and release operations bound range of program execution when it is safe to make copies

- Acquire: indicate that it is safe to make copies
  - Application guarantees synchronization is handled
  - Runtime removes all restrictions

- Release: indicate that copies are no longer allowed
  - Runtime flushes all dirty data back to original instance
  - Resumes enforcement of simultaneous restriction

- Apply to specific regions and fields
  - Dependence analysis performed just like other operations
Putting It All Together

Node 0
- FB
- CPU DRAM

- Copy i₁ -> i₃
- Acquire i₄
- Stencil
- Release i₄
- Copy i₁ -> i₃

Node 1
- CPU DRAM
- FB

- Copy i₂ -> i₄
- Acquire i₃
- Stencil
- Release i₃
- Copy i₂ -> i₄

Time
- Left
  - i₅
  - Owned
  - i₁
  - Left
  - i₄
- Right
  - Owned
  - i₂
  - Right
  - i₆

References:
- http://legion.stanford.edu

Date:
- December 4, 2014
A Note on Complexity

Isn’t this complex?
  Yes and No

What do we have to do today to get the same effect?
  MPI calls to move data between nodes
  MPI synchronization
  CUDA allocation and data movement
  Not composable

Legion approach is machine independent
  Simply specify coherence properties
  Where to use exclusive and simultaneous
  Where to perform acquires and releases
  Synchronization with phase barriers
  Can be composed hierarchically
S3D Example

- S3D is just a slightly more complex example
  - Lots of 1-D stencils
  - Done in 3-D space

- Explicit ghost regions
  - Fields for each stencil
  - PRF: 464 stencil fields

- Per field ghost regions and phase barriers
  - 8192 nodes: 4.5M barriers
  - Lots of messages in flight
  - Hide communication latency

December 4, 2014

http://legion.stanford.edu
Hierarchical Composition

What is the best way to write Legion programs?

- Both ways!

Implicit approach first
- Easier to write
- See how far it scales
- Might be enough

Explicit approach next
- Guarantees scalability

Compose them
- Enabled by hierarchical tasks and region trees
- Maps really well onto dragonfly topologies

Top-Level

SPMD Task

Node
Node

Node
Node

SPMD Task

Node
Node

Node
Node

SPMD Task

Node
Node

Node
Node

SPMD Task

Node
Node

Node
Node

Node
Node
Higher-Order Buffering

- Explicit ghost regions can be generalized
  - Double buffering
  - Triple buffering
  - ...

- Arbitrary depth to hide longer message latency
  - Who knows how bad exascale latency will be

- Code gets more complex
  - Build libraries
  - Have DSL compilers emit